Project 2 - VHDL simulation of MIPS FSM controller

Due May 09 (final deadline).

The purpose of this lab is to simulate the FSM controller for our MIPS processor, written in VHDL. A sample VHDL code for a simple FSM is available on the class web page. This is the second step in the process to design and simulate the entire processor. Next you will be asked to put the controller and the upgraded datapath together.

Project Description

Implement the FSM controller for multi-cycle MIPS processor described in Section 5.4 of the text. As before, we assume that Memory is external to our system. The inputs to your controller are coming from two fields of the Instruction Register (IR) already fetched from the memory: a 6-bit function field IR[31-26] and a 6-bit OPcode, IR[5-0]. All other fields of IR (rt, rs, rd) are controlling the datapath directly.

Specifically, you are asked to do the following:

1. Write a VHDL code for the FSM shown in Figure 5.42, slightly modified to account for our simplification: since our system does not include memory, our design does not include state 0. The initial state is state 1, “Instruction decode/register fetch”. The final states for each instruction (states 4, 5, 7, 8, and 9) will go back to state 1, instead of state 0. Use the names of the output signals exactly as they appear in the text.

Submit a hardcopy of the VHDL code, use small font and two-sided paper to save trees.

2. Simulate the VHDL code of the FSM using MaxPlus+ software for each instruction supported by our simplified processor: ADD, SUB, OR, AND, and a grossly simplified version of LW. Your version of instruction LW should fetch the data from Memory Data register (MD), instead of from Memory and write it to the register rd (in this case the other registers rs, rt are dummy). (Note: ORI, ANDI, real LW, SW, BEQ, JUMP cannot be fully simulated with Memory being external).

Specifically, for each instruction type, show the waveforms of the output control signals and demonstrate that the timing of those waveforms is correct. For example, the ADD instruction should generate three groups of control signals for the datapath in three clock cycles: cycle 1 = (ALUSrcA=0, ALUSrcB=11, ALUOp=00), cycle 2 = (ALUSrcA=1, ALUSrcB=00, ALUOp=10), cycle 3 = (RegDst=1, RegWrite=1, Memory=0), Convince yourself that these signals, when fed to the datapath will really control the execution of the ADD instruction (look what is happening in the datapath in response to those signals).

Examine the waveforms to make sure they are correct.

Submit waveforms for the ADD, SUB, AND, OR, and the simplified LW instructions. Again, keep them as small as possible while making them readable.